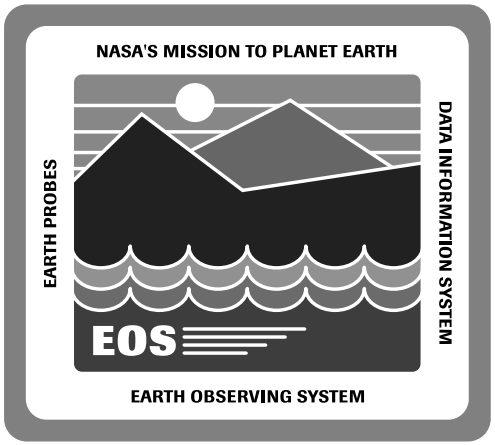
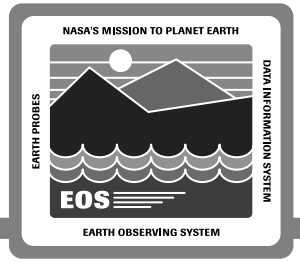


Interfaces to MSS

Yanamandra Sastry

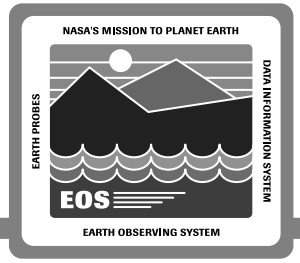
Developers Workshop
30 May 1995

Roadmap



- The MSS Role
- The Requirements
- How is Monitoring done?
- How is Control done?
- How is the instrumentation done?
- What does the application developer need to do?
- The MSS Header File
- The Development Scenario
- ECS Application Setup at Run-Time
- Run-Time Scenario - Event Logging
- Run-Time Scenario - Notification

The MSS Role

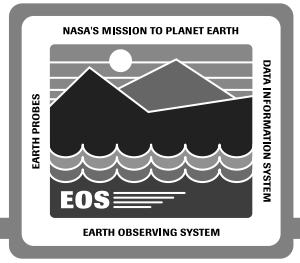


- **MSS is responsible for the management (monitor and control) of ECS resources (networks, systems and applications)**

- **What is monitored?**
  - **Health and status of ECS resources**
  - **Functional areas of Performance, Fault, Accountability & Security**

- **What is controlled?**
  - **Real-time configuration management of networks, systems and server applications**
    - **Startup**
    - **Shutdown**
    - **Suspend**
    - **Resume**

The Requirements



Four generic requirements for communications between MSS and ECS applications:

1) Event Logging - one way (from application to MSS)

- **Start/End of product generation**
- **Data Delivery Complete**

2) Notification - one way (to application from MSS)

- **Production string unavailable due to hardware error**

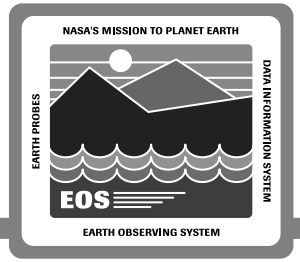
3) Lookup - bidirectional (called by application)

- **Retrieve User Profile for Order Shipping Address**

4) Instrumentation - bidirectional (called by MSS)

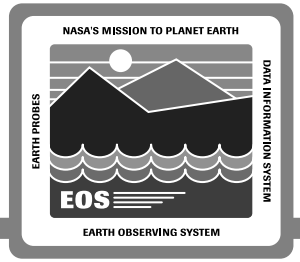
- **MSS has defined generic instrumentation**
  - **suspend, resume, etc**
- **Application developers need to work with MSS to help identify instrumentation specific to their application/subsystem**
  - **# orders processed, # browses, # searches, # products delivered**

How is monitoring done?



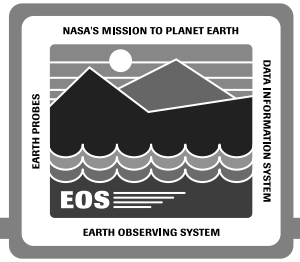
- **Monitoring is done in two ways:**
  - **By collecting information ABOUT managed objects**
    (Passive or non-intrusive data collection - no impact on developer)
    - **UNIX commands**
    - **Other utilities**
  - **By collecting information FROM managed objects**
    (Active or intrusive data collection - active developer participation)
    (This is static, and must be developed a-priori)
    - **Accomplished via instrumentation of the managed object**
    - **Applications must be developed to:**
      - **Provide information on events that generate management data**
      - **Respond to instrumentation (generic & specific) requests from MSS**
      - **COTS need to be "wrapped" to provide control**

How is control done?



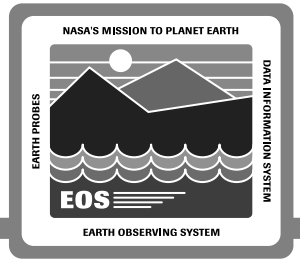
- Accomplished via instrumentation
- Application must be instrumented to respond to control requests
- COTS need to be “wrapped” to provide control
- The interface to MSS is consistent in either case

How is the instrumentation done?



- Needs active participation of the developers
- MSS will define generic instrumentation
  - suspend, resume, etc
- Developers need to define specific instrumentation
  - # orders, # searches, # browses, etc
- What is available from MSS?
  - A header file, “MSSOBJECT.H”, to include in source file for application development
  - An object library, “MSSOBJECT.O”, to link with, for routines MSS will provide
  - Insulates the developer from changes in the underlying MSS protocol

What do developers need to do?

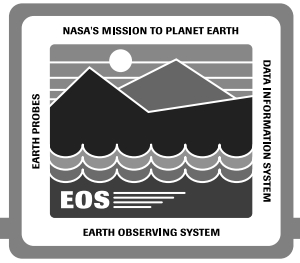


- Define/Identify special instrumentation where required
- Work with MSS to develop the information model
  - e.g., dependencies/rules for startup/restart
- Supply action routines to be invoked in response to events
- Based on this, MSS will provide the class definitions
- Use MSS-provided classes for necessary instrumentation
- Details on the MSS classes to follow

ASSUMPTIONS:

- OODCE is available on all ECS managed hosts
- All servers being developed are OODCE servers

The MSS Header File

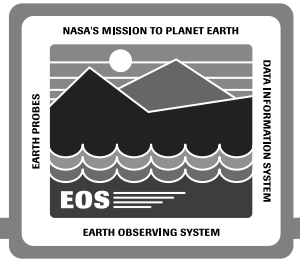



```
Class MSSObject
{
   public:
void MSSEvent(.....);
void MSSNotify(char*,...);
void ApplicationRequestAndResponse(char*...);
long AgentRequestAndResponseInt(...);
char*  AgentRequestAndResponseStr(char*, ...);

void SetNotifyCallBack( void (* NotifyCallBack)(char *));
void SetInstrumentationCallBackString(char * (*InstrumentationCallBackStr)(char *
));
void SetInstrumentationCallBackInteger(long (*InstrumentationCallBackInt)(char*));

}
```

The Development Scenario



1) Write the callback functions for Notifications and Instrumentation. These need to be jointly developed with MSS since they are shared.

2) Instantiate the MSS Object

3) Register the callback functions with the MSSObject

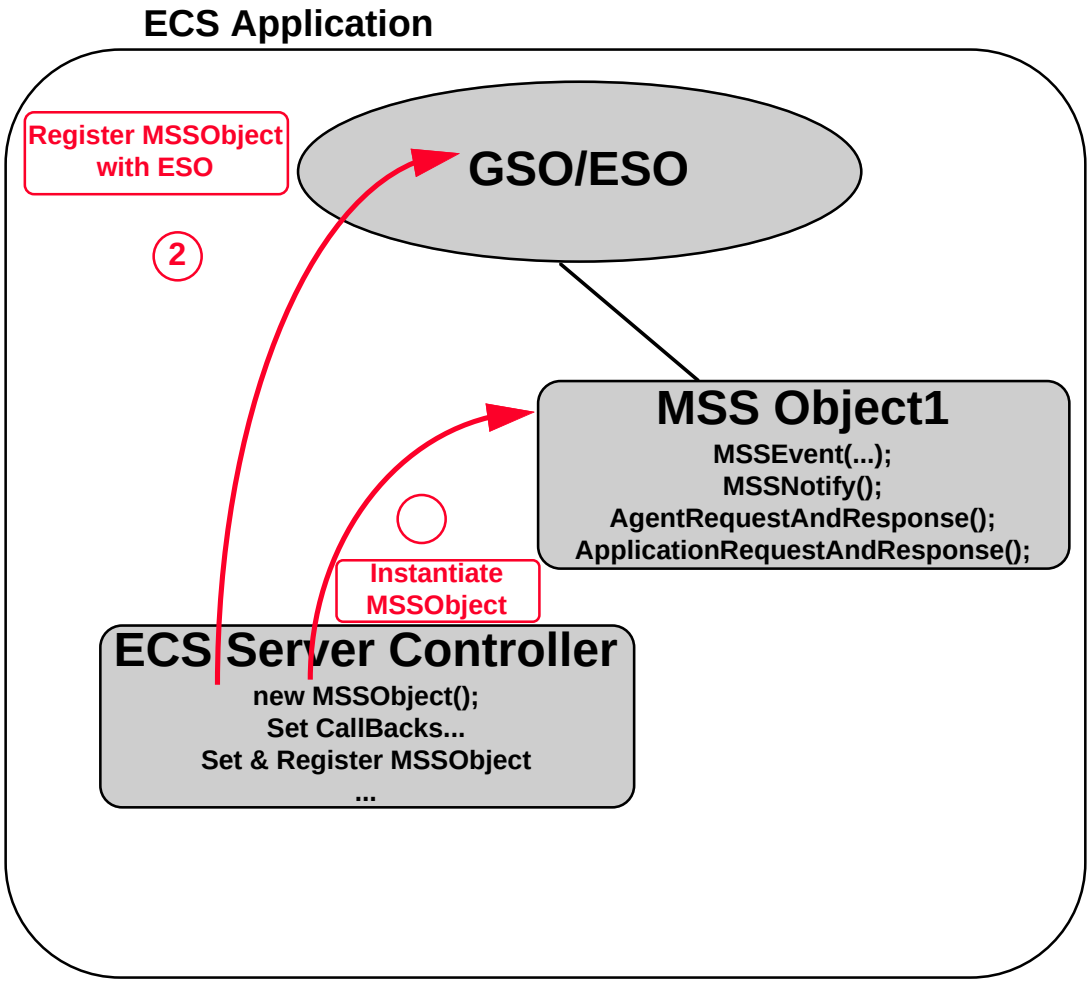
4) Register the MSSObject with ESO (specialized GSO)

...

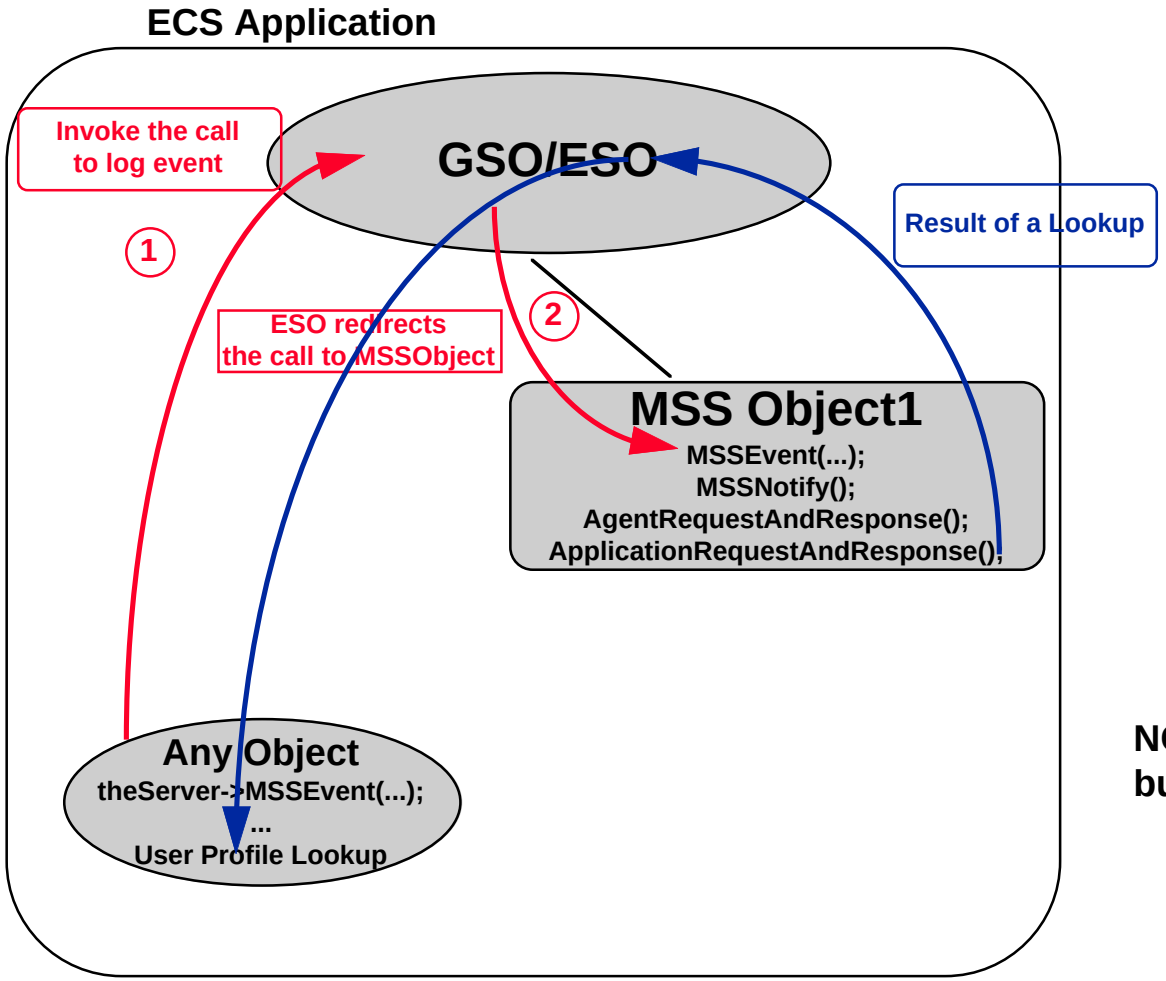
...

n) Listen on the management interface for MSS queries

ECS Application setup at run-time

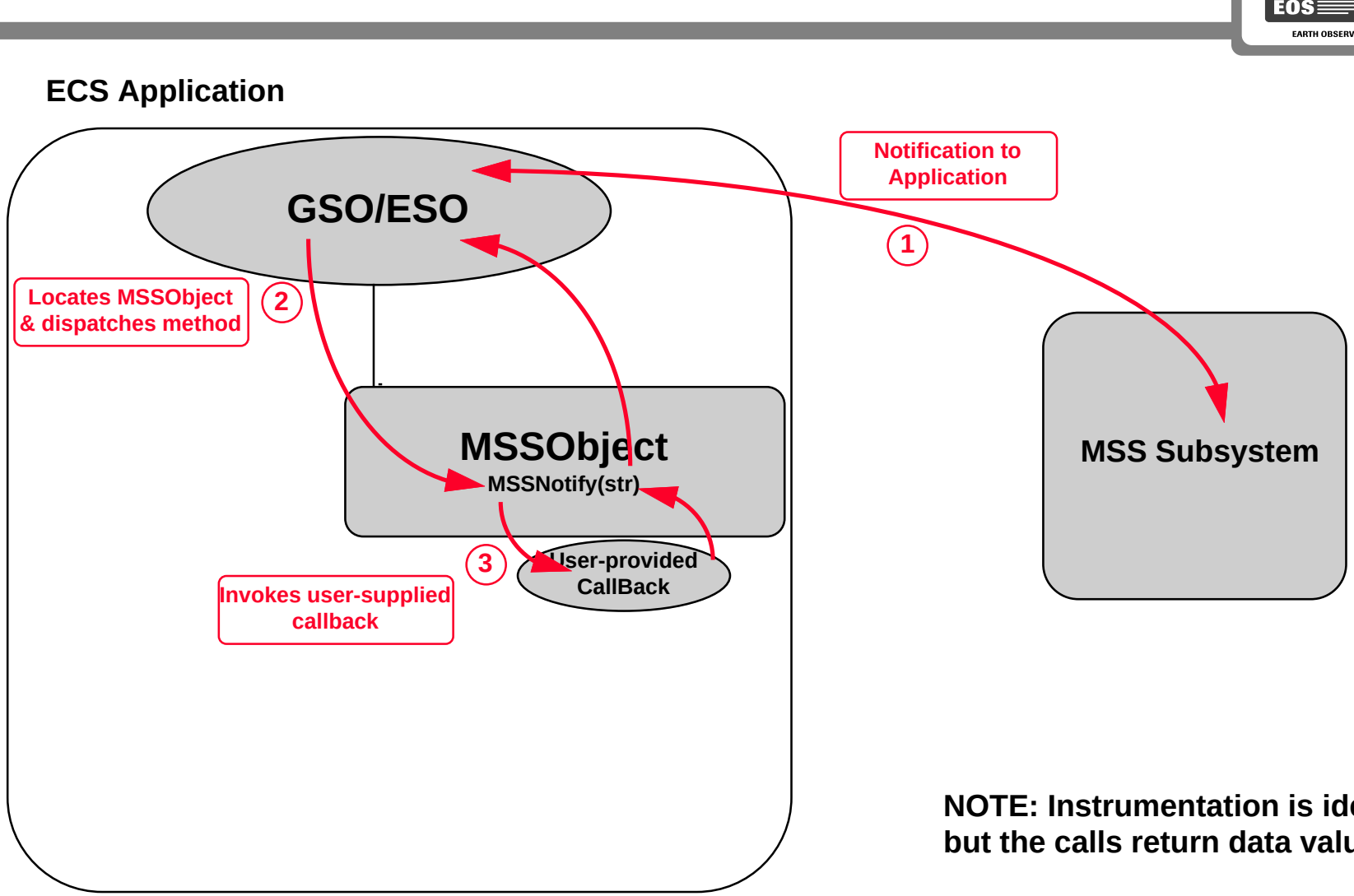


Run-Time Scenario - Event Logging



NOTE: Lookup is identical, but the call returns data values

Run-Time Scenario - Notification



NOTE: Instrumentation is identical, but the calls return data values